

N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM
MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT
CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED
IN THE INTEREST OF MAKING AVAILABLE AS MUCH
INFORMATION AS POSSIBLE

A Computer Program for Determining Truncation Error Coefficients for Runge-Kutta Methods

(NASA-TM-81143) A COMPUTER PROGRAM FOR
DETERMINING TRUNCATION ERROR COEFFICIENTS
FOR RUNGE-KUTTA METHODS (NASA) 27 p
HC A03/MF A01

N80-30095

CSSL 12A

Unclas
G3/64 26923

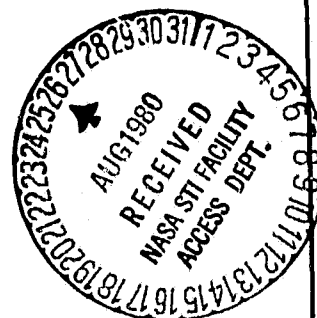
Mission Planning and Analysis Division

July 1980

NASA

National Aeronautics and
Space Administration

Lyndon B. Johnson Space Center
Houston, Texas



80-FM-13

JSC-16429

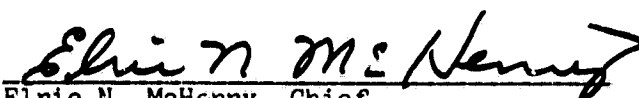
SHUTTLE PROGRAM

A COMPUTER PROGRAM FOR DETERMINING
TRUNCATION ERROR COEFFICIENTS
FOR RUNGE-KUTTA METHODS

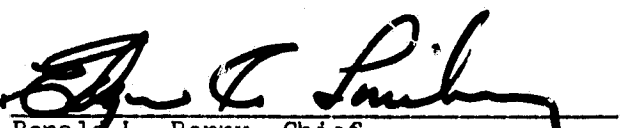
By M. Kathleen Horn*

JSC Task Monitor: Victor Bond
Software Development Branch

Approved:


Elric N. McHenry, Chief
Software Development Branch

Approved:


Ronald L. Berry, Chief
Mission Planning and Analysis Division

Mission Planning and Analysis Division
National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

July 1980

*National Research Council Research Associate

CONTENTS

Section		Page
1.0	<u>INTRODUCTION</u>	1
2.0	<u>THE GENERATION OF TRUNCATION ERROR COEFFICIENTS</u>	3
3.0	<u>DESCRIPTION OF THE FORTRAN PROGRAM</u>	6
4.0	<u>CONCLUDING REMARKS</u>	8
5.0	<u>REFERENCES</u>	9
APPENDIX A - SUBROUTINE RKEQ		A-1

TABLES

Table		Page
I	DIMENSIONING PARAMETER, LS, FOR ORDERS 1 THROUGH 12	7

1.0 INTRODUCTION

The solution of the initial value problem for ordinary differential equations (ODE's)

$$\frac{dy}{dt} = f(t,y), \quad y(t_0) = y_0 \quad (1)$$

may be treated by several different numerical methods. Runge-Kutta (RK) algorithms are a type of method well suited to solving equation (1) for many classes of functions, f , because of their simplicity and their accuracy. The RK algorithm is derived using a direct comparison with a truncated Taylor series, giving the accuracy of the Taylor series without the difficulty of determining complicated partial derivatives. The comparison between the Taylor series expansion of the solution vector and the solution determined by the RK algorithm results in a number of expressions referred to as truncation error coefficients, $T_{i,j}$. Associated with each term of order i in the Taylor series (or with each power of h , the integration stepsize) are λ_i truncation error coefficients. For an RK algorithm to be of order p , the $T_{i,j}$ coefficients must be identically zero for $i = 1, \dots, p$; $j = 1, \dots, \lambda_i$. These vanishing truncation error coefficients are referred to as equations of condition. The nonvanishing error coefficients, however, are of equally great importance since they indicate how closely the RK solution approximates a Taylor series solution of higher order. The equations of condition determine the validity of an RK algorithm; the nonvanishing error coefficients explain the differences between particular RK algorithms of the same order. While a user may apply an RK algorithm, never considering the truncation error coefficients, awareness of the effect of these terms is important both in the selection of a specific algorithm and in the analysis of difficulties encountered during the solution of a particular ODE.

D. G. Bettis,^a has developed an algorithm for generating the truncation error coefficients for RK methods designed to treat systems of both first- and second-order ODE's directly. The recursive nature of this algorithm lends itself readily to computer programming, generating high order error coefficients with little added difficulty. Such an algorithm, implemented in a numerical code, is an essential tool for anyone developing coefficients for RK algorithms and is of interest to the user of RK methods in analyzing the effectiveness of specific RK algorithms. A Fortran subroutine, RKEQN, written to accompany reference 1, generated the truncation error coefficients through order 10 but required a great amount of storage location, particularly when a double precision version of the program was needed. The basic structure of this original program has been reformulated to reduce storage requirements significantly and to accommodate variable dimensioning. This new Fortran program, SUBROUTINE RKEQ, determines truncation error coefficients for RK algorithms in the sequence presented in reference 1 for orders 1 through 10 and extends the order of coefficients

^aFrom a private communication with D. G. Bettis, 1978.

through 12 with the 11th- and 12th-order terms determined following the patterns used to establish the lower order coefficients. Both subroutines (RKEQN and RKEQ) are also written to treat RK m-fold methods (refs. 2 and 3) which utilize m known derivatives of f to increase the order of the algorithm. Setting m = 0 gives the classical RK algorithm.

2.0 THE GENERATION OF TRUNCATION ERROR COEFFICIENTS

The solution of equation (1) at $t_1 = t_0 + h$, using the RK algorithm, is written

$$y_1 = y_0 + \sum_{k=0}^p c_k f_k \quad (2)$$

where

$$f_0 = f(t_0, y_0)$$

$$f_k = f(t_0 + \alpha_k h, y_0 + h \sum_{\lambda=0}^{k-1} \beta_{k,\lambda} f_\lambda)$$

where $p + 1$, the number of evaluations of f computed, is referred to as the number of stages. The truncation error coefficients, $T_{i,j}$, determined by comparing the Taylor series expansion of equation (2) with the Taylor series expansion of the solution about t_0 , are nonlinear combinations of the C , α , and β coefficients. For the classical RK algorithm, the j th error coefficient of order i assumes the form

$$T_{i,j} = \{F_{i,j} - 1/(pA_{i,j})\} / B_{i,j} \quad (3)$$

$j = 1, \dots, \lambda_i$ where $p = i$. For the m -fold algorithm, $p = m + 1$ and corresponds to the order of the term. (For $m = 0$, the m -fold algorithm is identical to the classical RK formula.) The $A_{i,j}$ and $B_{i,j}$ terms are constants (or functions of m for m -fold methods) and may be determined by recursive relations. (One should note that while references to m -fold RK algorithms may appear to complicate matters, the inclusion of these methods in RKEQ (or RKEQN) involves the insertion of only a few additional lines of coding. Once these additions are made, the classical RK error coefficients and the m -fold error coefficients are determined identically.) The complicated expression to generate in equation (3) is the $F_{i,j}$ term, which is a combination of the C , α , and β coefficients

$$F_{i,j} = \sum_{k=k_0}^p c_k S_{i,j,k} \quad (4)$$

with $S_{i,j,k}$ being a combination of α and β coefficients and where k_0 depends upon the number of summations embedded in $S_{i,j,k}$.

The algorithm developed by Bettis and used for generating these $S_{i,j,k}$, $A_{i,j}$, and $B_{i,j}$ terms is documented in reference 1, where the $S_{i,j,k}$ terms are written in an abbreviated notation, e.g., $S_{8,13} = \alpha^2 \beta \alpha \beta \beta \alpha$, with the subscripting and embedded summations being suppressed. The rules for writing the entire $S_{i,j,k}$ terms are also described thoroughly in reference 1. For the sake of interpreting the program, however, a few features need to be known about generating the terms in abbreviated notation. Denoting the number of truncation error coefficients of order μ , by λ_μ , and suppressing the k subscript, the first $2\lambda_{\mu-1}$ $S_{i,j,k}$ expressions are generated from the $S_{i-1,j,k}$ terms. The remaining $\lambda_\mu - 2\lambda_{\mu-1}$ expressions, referred to as composite sums, are formulated as products of lower order S terms. The $A_{i,j}$ and $B_{i,j}$ constants are also generated from simple relationships involving previous A and B terms. In generating the first λ_{i-1} terms of order i , the $S_{i-1,j}$ expressions are premultiplied by an α . (Adjacent α 's represent actual multiplication.) Thus, $S_{9,13} = \alpha S_{8,13}$, $S_{13} = \alpha^3 \beta \alpha \beta \beta \alpha$. The next λ_{i-1} terms, $S_{i,j}$, $j = \lambda_{i-1} + 1, \dots, 2\lambda_{i-1}$ are generated by premultiplying the $S_{i-1,j-\lambda_{i-1}}$ expressions by a β , e.g., $S_{9,128} = \beta S_{8,13} = \beta \alpha^2 \beta \alpha \beta \beta \alpha$. (Adjacent β 's do not represent multiplication of the β coefficients since a summation sign precedes each β when the entire S term is written, e.g.,

$$S_{6,4,k} = \alpha^2 \beta \beta \alpha = \alpha_k^2 \sum_{\ell=2}^{k-1} \beta_{k,\ell} \sum_{m=1}^{\ell-1} \beta_{\ell,m} \alpha_m \quad (5)$$

($S_{6,4,k} \neq \alpha^2 \beta^2 \alpha$.) The recursion relations, then, for the first λ_{i-1} terms of order i , ($i > 2$), are

$$S_{i,j} = \alpha S_{i-1,j}$$

$$A_{i,j} = A_{i-1,j}$$

$$B_{i,j} = \mu B_{i-1,j}$$

for $j = 1, 2, \dots, \lambda_{i-1}$, where μ is the power of the leading α in the $S_{i-1,j}$ term, and

$$S_{i,j} = \beta S_{i-1,j-\lambda_{i-1}}$$

$$A_{i,j} = (m+i-1) A_{i-1,j-\lambda_{i-1}}$$

$$B_{i,j} = B_{i-1,j-\lambda_{i-1}}$$

for $j = \lambda_{i-1} + 1, \dots, 2\lambda_{i-1}$. ($S_{1,1} = 1$, and $S_{2,1} = \alpha$.) The first λ_{i-1} S expressions are referred to as alpha terms in subroutine RKEQ, while the next λ_{i-1} expressions are called beta terms. The remaining terms, composite sums, are generated by considering the weight factors of the S terms. The $S_{\mu,j}$ expressions have a weight factor of $\mu-1$ (i.e., the number of α and β coefficients included in the S term). The composite sums of order i are all products of $S_{\mu,j}$ terms having initial β coefficients, whose weight factors add up to $i-1$. Subroutine RKEQ determines these composite sums in a separate block of the subprogram, calling subroutine CROSS to perform the multiplication of the S, A, and B terms. The $A_{i,j}$ and $B_{i,j}$ terms of a composite sum are the products of the A and B constants whose corresponding S terms form the composite sum. (When an S term is raised to the power k , an additional $k!$ multiplies the B constant.)

3.0 DESCRIPTION OF THE FORTRAN PROGRAM

Subroutine RKEQ determines the truncation error coefficients (TEC) for a given set of RK coefficients and returns TERROR,

$$\text{TERROR} = \left\{ \sum_{j=1}^{\lambda_1} T_{1,j}^2 \right\}^{1/2}$$

for a specified order i . Since RK algorithms with embedded pairs of solutions, e.g. RK-Fehlberg formulas, are often studied, RKEQ is written to treat two algorithms simultaneously, which use identical α and β coefficients but

different C_k and \hat{C}_k coefficients. (TERROR is formed using the \hat{C}_k coefficients.) The Greek letters α_i and β_i are replaced by A(I) and B(I) and the $A_{i,j}$ and $B_{i,j}$ constants are denoted AA(I,J) and BB(I,J), respectively.

The input parameters for RKEQ are:

(1) The RK coefficients

- (a) A(K) α_k , the alpha coefficients
- (b) CO, C(K) C_0, C_k , the C_k coefficients for the first solution
- (c) CHO, CH(K) \hat{C}_0, \hat{C}_k , the \hat{C}_k coefficients for the second solution used to form TERROR
- (d) BO(K), B(K,L) $\beta_{k,0}, \beta_{k,l}$, the beta coefficients

where $K = 1, 2, \dots, R$, $L = 1, 2, \dots, K - 1$, R an integer with $R + 1$ being the number of stages of the algorithm, and

(2) The integers controlling orders and options

- (a) R = the index for dimensioning the RK coefficients
- (b) IORDER = the maximum order to be treated
- (c) ITERR = the order of TEC used to form TERROR
- (d) IOPT = the options for operating the program. For IOPT = 1, RKEQ computes and prints all TEC(I,J) for $I = 1, \dots, \text{IORDER}$. For IOPT = 2, RKEQ computes and prints TEC(IORDER, J) only. For IOPT = 3, RKEQ computes but does not print TEC(ITERR, J). (For all options TERROR is computed, which may require internal adjustments to the order.)
- (e) MFOLD = an integer giving the number of known derivatives of f . For the classical RK algorithm, MFOLD = 0.

(f) LS = a dimensioning index for the work arrays, S, AA and BB.

The output parameter for RKEQ is ERROR, the Euclidean norm of the TEC of order ITERR. Depending upon the option used, RKEQ may print values of TEC, but these are not returned to the main program.

Parameters S, AA, and BB are used internally by RKEQ to compute the TEC terms. To take advantage of variable dimensioning, these parameters are given in the calling sequence with dimensions S(LS,R), AA(LS), BB(LS). The RK coefficients should be dimensioned A(R), C(R), CH(R), B(R, R), BO(R), R an integer, where $R + 1$ is the number of stages for the algorithm.

The calling sequence for RKEQ is

```
SUBROUTINE RKEQ(A, C, CO, CH, CHO, B, BO, R, IORDER, ITERR, IOPT, MFOLD,
  ERROR, S, LS, AA, BB).
```

which, if a printing option is used, will give the TEC from the C solution in the first column and the TEC from the CH solution in the second column. Integers IORDER, ITERR, and IOPT are reset within the subroutine, and any adjustments made to protect against exceeding dimension or option limits are made to the new variables, so that the user may enter constant values in the calling sequence of the driving program.

A sample calling sequence for a six-stage, fifth-order algorithm is

```
CALL RKEQ(A, C, CO, CH, CHO, B, BO, 5, 7, 6, 1, 0, ERROR, S, 48, AA, BB)
```

which computes and prints all TEC through order 7 for the classical RK formulas, using the 6th-order terms of the CH solution to form ERROR. Using the calling sequence

```
CALL RKEQ(A, CH, CHO, C, CO, B, BO, 5, 7, 6, 1, 0, ERROR, S, 48, AA, BB)
```

generates similar information except that ERROR is formed by the C solution

(and the \hat{C} TEC terms are now printed in the first column.)

The minimum value of LS for a given ORDER, I, is found in table I. A listing of subroutines RKEQ and CROSS may be found in the appendix.

TABLE I.- DIMENSIONING PARAMETER, LS, FOR ORDERS 1 THROUGH 12

ORDER	1	2	3	4	5	6	7	8	9	10	11	12
LS _{MIN}	1	1	2	4	9	20	48	115	286	719	1842	4766

4.0 CONCLUDING REMARKS

A program, which evaluates the truncation error coefficients, is an essential tool in the development of Runge-Kutta algorithms and in the comparison of existing RK algorithms. By structuring the routine in the given form, a substantial savings in storage occurs in generating these truncation error coefficients using the recursive formulation presented by D. G. Bettis (ref. 1). The extension to orders higher than 12 is relatively simple but not of great practical use at the present time.

REFERENCES

1. Bettis, D. G.; and Horn, M. K.: Computation of Truncation Error Terms for Runge-Kutta Methods. TICOM Report 77-14, December 1977.
2. Fehlberg, E.: New High-Order Runge-Kutta Formulas with Step-size Control for Systems of First- and Second-Order Differential Equations. ZAMM, Vol. 44, 1964.
3. Fehlberg, E.: New High-Order Runge-Kutta Formulas with an Arbitrarily Small Truncation Error. ZAMM, Vol. 6, 1966.

80FM13

APPENDIX A
SUBROUTINE RKEQ

```

SUBROUTINE RKEQ(A,C,CO,CH,CHO,B,BO,R,
1 IORDER,ITERR,IOPT,MFOLD,TERROR,S,LS,AA,BB)
INTEGER R,ORDER,OPTION
DOUBLE PRECISION A(R),C(R),CH(R),B(R,R),BO(R)
DOUBLE PRECISION CO,CHO
DOUBLE PRECISION S(LS,R),AA(LS),BB(LS)
DOUBLE PRECISION SUM1,SUM3,X1,ZERO,UNITY,TWO,
1 MP(12),MM(12),FACT(12)
DOUBLE PRECISION P,PP1,PM1,PM2,ZAPP,TERROR
DIMENSION M(12),LIMIT(12),INDEX(11)
LOGICAL EVEN

C
DATA LIMIT/1,1,2,4,9,20,48,115,286,719,1842,4766/
DATA INDEX/1,1,2,5,11,28,67,171,433,1123,2924/
DATA ZERO,UNITY,TWO,ZAPP/0.0D0,1.0D0,2.0D0,1.0D-14/
DATA MAXORD/12/

C
C*****
C
DATA KPRINT/0/

C
C*****
C
C
C
C
C
C
SUBROUTINE RKEQ IS A FORTRAN SUBROUTINE WRITTEN BY
M.K. HORN WHICH IMPLEMENTS THE ALGORITHM DEVELOPED
BY D.G. BETTIS TO GENERATE TRUNCATION ERROR COEFFI-
CIENTS FOR RUNGE-KUTTA ALGORITHMS.

REFERENCE: BETTIS,D.G. AND M.K. HORN, 'COMPUTATION
OF TRUNCATION ERROR TERMS FOR RUNGE-KUTTA METHODS,'
TICOM REPORT 77-14, DECEMBER, 1978.

SUBROUTINE RKEQ DETERMINES THE TRUNCATION ERROR
COEFFICIENTS (TEC) FOR A RUNGE-KUTTA ALGORITHM HAVING
AN EMBEDDED PAIR OF SOLUTIONS

R

Y = Y + H SUM C F
1 0 K K
K=0

R

YH = Y + H SUM CH F
1 0 K K

```


C	K=0	00005000
C		00005100
C		00005200
C		00005300
C	WHERE $F_0 = F(T_0, Y_0)$	00005400
C		00005500
C		00005600
C		00005700
C		00005800
C		00005900
C	$F_K = F(T_0 + A_K H, Y_0 + H * \sum_{L=0}^{K-1} B_{K,L} F_L)$	00006000
C		00006100
C		00006200
C	FOR MFOLD = 0, RKEQ FORMS THE TEC FOR THE CLASSICAL RK	00006300
C	FORMULAS OF ORDER = IORDER. FOR MFOLD = M, RKEQ FORMS	00006400
C	THE TEC FOR MFOLD-RK FORMULAS OF ORDER = M + IORDER.	00006500
C		00006600
C		00006700
C		00006800
C	-----	00006900
C	OPTION .EQ. 1 COMPUTES AND PRINTS ALL TRUNCATION	00007000
C	ERROR COEFFICIENTS THROUGH ORDER =	00007100
C	IORDER	00007200
C	OPTION .EQ. 2 COMPUTES AND PRINTS ONLY T.E.C. OF	00007300
C	ORDER = IORDER	00007400
C	OPTION .EQ. 3 COMPUTES T.E.C. AS IN OPTION = 2 BUT	00007500
C	DOES NOT PRINT	00007600
C		00007700
C		00007800
C	-----	00007900
C		00008000
C		00008100
C	ADJUSTS INPUT PARAMETERS IF THESE ARE NOT WITHIN	00008200
C	ALLOWABLE RANGE	00008300
C		00008400
C		00008500
C	ORDER = IORDER	00008600
C	LIMITS = ITERR	00008700
C	IF (ORDER .LT. LIMITS) ORDER = LIMITS	00008800
C	IF (LS. LT. LIMIT(MAXORD)) GO TO 1	00008900
C	IF (ORDER .LE. MAXORD) GO TO 1	00009000
C	ORDER = MAXORD	00009100
C	PRINT 507,MAXORD	00009200
C	507 FORMAT(52H ORDER REQUESTED IS BEYOND CAPABILITY OF THE PROGRAM	00009300
C	1 ,/,24H ORDER LOWERED--ORDER = ,I2)	00009400
C	IF (ITERR .LE. ORDER) GO TO 1	00009500
C	LIMITS = ORDER	00009600
C	PRINT 526,LIMITS	00009700
C	526 FORMAT(45H ITERR REQUESTED IS LARGER THAN MAXIMUM ORDER	00009800
C	1 ,/,35H ITERR HAS BEEN REDUCED TO ORDER = ,I2)	00009900

C		00010000
	1 CONTINUE	00010100
C		00010200
	TERROR = ZERO	00010300
	OPTION = IOPT	00010400
	IF (IOPT .LE. 0 .OR. IOPT .GT. 3) OPTION = 1	00010500
	IF (LS .GE. LIMIT(ORDER)) GO TO 3	00010600
	2 CONTINUE	00010700
	ORDER = ORDER - 1	00010800
	IF (LS .LT. LIMIT(ORDER)) GO TO 2	00010900
	PRINT 505,ORDER,IORDER,LIMIT(IORDER)	00011000
	505 FORMAT(50H THE ORDER SPECIFIED HAS BEEN REDUCED TO ORDER =	00011100
	1 ,I3,/,32H BECAUSE OF INSUFFICIENT STORAGE ,/,	00011200
	2 14H FOR ORDER = ,I3,20H LS MUST BE .GE. ,I5,	00011300
	3 /,56H TERROR = SQRT(SUM(T.E.C.(I,J)*T.E.C.(I,J))) IS COMPUTED	00011400
	4 ,/,14H FOR I = ORDER)	00011500
C		00011600
	IF (LIMITS .LE. ORDER) GO TO 3	00011700
	LIMITS = ORDER	00011800
	PRINT 527,LIMITS	00011900
	527 FORMAT(45H ITERR REQUESTED IS LARGER THAN THE PROVIDED	00012000
	1,/,50H DIMENSIONING. ITERR HAS BEEN REDUCED TO ITERR = ,	00012100
	2 I2)	00012200
	3 CONTINUE	00012300
C		00012400
C		00012500
C		00012600
	IF (ORDER .GT. MAXORD) ORDER = MAXORD	00012700
C		00012800
C		00012900
C	-----	00013000
C		00013100
C	MP(J) = (MFOLD+J) D.P.	00013200
C	M(J) = J INTEGER	00013300
C	FACT(J) = J D.P.	00013400
C	MM(J) = (MFOLD+J) D.P.	00013500
C		00013600
C	INDEX--COUNTS THE NUMBER OF S(J,K) WITH A GIVEN A**K	00013700
C	AS THE FIRST TERM IN THE EXPRESSION, E.G.,	00013800
C	FOR J=7, THERE ARE 1 A**6, 1 A**4, 2 A**3,	00013900
C	4 A**2, AND 9 A**1 @S	00014000
C		00014100
C	-----	00014200
C		00014300
	X1 = A(1) - B0(1)	00014400
	JJ = 1	00014500
	550 FORMAT(15H ERROR IN BETA(,I2,10H) SUM = ,D15.7)	00014600
	IF (DABS(X1) .GE. ZAPP) PRINT 550,JJ,X1	00014700
	DO 5 J = 2,R	00014800
	X1 = A(J) - B0(J)	00014900

	JLOW = J-1	00015000
	DO 4 K = 1,JLOW	00015100
	4 X1 = X1 - B(J,K)	00015200
	IF (DABS(X1) .GE. ZAPP) PRINT 550,J,X1	00015300
	5 CONTINUE	00015400
C		00015500
	IF (KPRINT .EQ. 1) READ PRINT 499	00015600
C		00015700
C		00015800
C	SETS VALUES OF FACTORIALS USED IN THE PROGRAM.	00015900
C	ADJUSTMENTS IN THE PROGRAM TO ACCOMODATE M-FOLD	00016000
C	RUNGE-KUTTA ALGORITHMS OCCUR HERE. IF MFOLD = 0,	00016100
C	THE CLASSICAL RUNGE-KUTTA T.E.C. OCCUR	00016200
C		00016300
C		00016400
	IF (MFOLD .GT. 0) GO TO 6	00016500
	EVEN = .TRUE.	00016600
	GO TO 8	00016700
	6 CONTINUE	00016800
	N1 = MFOLD / 2	00016900
	N2 = 2 * N1	00017000
	EVEN = .FALSE.	00017100
	IF (N2 .EQ. MFOLD) EVEN = .TRUE.	00017200
	8 CONTINUE	00017300
	MP(1) = DFLOAT(MFOLD)+UNITY	00017400
	M(1) = 1	00017500
	X1 = UNITY	00017600
	FACT(1) = UNITY	00017700
	DO 10 I = 2,ORDER	00017800
	X1 = X1 + UNITY	00017900
	FACT(I) = FACT(I-1)*X1	00018000
	MP(I) = MP(I-1) + UNITY	00018100
	10 M(I) = M(I-1) + 1	00018200
	M2 = 1	00018300
	M1 = 0	00018400
	11 CONTINUE	00018500
	M1 = M1 + 1	00018600
	M2 = M2*M1	00018700
	IF (M1 .LT. MFOLD+1) GO TO 11	00018800
	MM(1) = DFLOAT(M2)	00018900
	DO 12 I = 2,ORDER	00019000
	12 MM(I) = MP(I)* MM(I-1)	00019100
C		00019200
C		00019300
	IF (KPRINT .EQ. 1) PRINT 499	00019400
C		00019500
C	SETS INITIAL VALUES OF S(I,J) EQUAL TO ZERO	00019600
C	AND SETS AA(J) AND BB(J) EQUAL TO UNITY	00019700
C		00019800
	DO 14 J = 1,LS	00019900

AA(J) = UNITY	00020000
BB(J) = UNITY	00020100
DO 14 K = 1,R	00020200
14 S(J,K) = ZERO	00020300
C	00020400
IF (OPTION .GT. 1 .AND. LIMITS .NE. 1) GO TO 21	00020500
C	00020600
C	00020700
EVALUATES T.E.C. OF ORDER 1	00020800
C	00020900
SUM1 = UNITY/DFLOAT(MFOLD + 1) - CO	00021000
SUM3 = UNITY/DFLOAT(MFOLD + 1) - CHO	00021100
DO 20 I = 1,R	00021200
SUM1 = SUM1 - C(I)	00021300
SUM3 = SUM3 - CH(I)	00021400
20 CONTINUE	00021500
C	00021600
JJ1 = 1	00021700
IF (LIMITS .EQ. 1) TERROR = SUM3*SUM3	00021800
IF (OPTION .EQ. 3) GO TO 21	00021900
PRINT 500,JJ1	00022000
PRINT 501,JJ1,JJ1,SUM1,SUM3	00022100
500 FORMAT(36H TRUNCATION ERROR TERMS X-ORDER = ,I2,2X	00022200
1 ,//)	00022300
501 FORMAT(2(2X,I4),2(D15.7))	00022400
C	00022500
21 CONTINUE	00022600
C	00022700
C	00022800
SETS S(1,J) TERMS	00022900
C	00023000
KOUNT = 1	00023100
C	00023200
DO 26 I = 1,R	00023300
IF (MFOLD .GT. 0) GO TO 22	00023400
S(1,I) = A(I)	00023500
GO TO 26	00023600
22 CONTINUE	00023700
IF (EVEN) GO TO 24	00023800
S(1,I) = DABS(A(I))*MP(1)	00023900
GO TO 26	00024000
24 CONTINUE	00024100
X1 = DABS(A(I))*MP(1)	00024200
S(1,I) = DSIGN(X1,A(I))	00024300
26 CONTINUE	00024400
C	00024500
C	00024600
EVALUATES T.E.C. FOR ORDER = KOUNT = 2,3	00024700
C	00024800
AA(1) = UNITY	00024900
BB(1) = MM(1)	
28 CONTINUE	

P = TWO + DFLOAT(MFOLD)	00025000
PP1 = P+UNITY	00025100
30 CONTINUE	00025200
IF (LIMITS .EQ. KOUNT) JJ1 = KOUNT - 1	00025300
IF (OPTION .GT. 1 .AND. LIMITS .NE. KOUNT) GO TO 34	00025400
JJ2 = 0	00025500
JJ1 = KOUNT + 1	00025600
PRINT 500,JJ1	00025700
DO 33 K = 1,KOUNT	00025800
JJ2 = JJ2 + 1	00025900
SUM1 = UNITY/(AA(K)*P)	00026000
SUM3 = SUM1	00026100
DO 32 I = 1,R	00026200
SUM1 = SUM1 - C(I)*S(K,I)	00026300
SUM3 = SUM3 - CH(I)*S(K,I)	00026400
32 CONTINUE	00026500
IF (LIMITS .EQ. KOUNT) TERROR = TERROR + SUM3*SUM3	00026600
IF (OPTION .EQ. 3) GO TO 33	00026700
PRINT 501,JJ1,JJ2,SUM1,SUM3	00026800
33 CONTINUE	00026900
C	00027000
34 CONTINUE	00027100
C	00027200
IF (KOUNT .EQ. 2) GO TO 38	00027300
C	00027400
C	00027500
C	00027600
KOUNT = 2	00027700
P = P + UNITY	00027800
PP1 = PP1 + UNITY	00027900
DO 35 I = 2,R	00028000
S(2,I) = ZERO	00028100
IM1 = I-1	00028200
DO 35 J = 1,IM1	00028300
35 S(2,I) = S(2,I) + B(I,J)*S(1,J)	00028400
DO 36 I = 1,R	00028500
36 S(1,I) = S(1,I)*A(I)	00028600
AA(2) = TWO	00028700
BB(1) = MM(2)	00028800
BB(2) = MM(1)	00028900
GO TO 30	00029000
38 CONTINUE	00029100
IF (ORDER .LE. 3) GO TO 182	00029200
LIM1 = 3	00029300
C	00029400
C	00029500
C	00029600
EVALUATES T.E.C. FOR ORDERS GREATER THAN THREE	00029700
DO 180 J = 4,ORDER	00029800
P = P + UNITY	00029900
PP1 = P + UNITY	00029900

	PM1 = P - UNITY	00030000
	PM2 = P - TWO	00030100
	IF (KPRINT .EQ. 1) READ 497,II	00030200
	LIM1 = LIM1 + 1	00030300
	LIMA = LIMIT(J-1)	00030400
	LIMB = LIMA	00030500
C		00030600
C	COMPUTES S(1,I)--ALL OTHER S(J,K) TERMS INVOLVING A	00030700
C	LEADING ALPHA ARE ALREADY DETERMINED	00030800
C	EXCEPT FOR THE POWER OF ALPHA WHICH	00030900
C	IS DETERMINED BY IN INDEX AND J	00031000
C		00031100
C	AA(1) = AA(1)	00031200
	BB(1) = BB(1) * MP(LIM1-1)	00031300
	DO 42 I = 1,R	00031400
	42 S(1,I) = S(1,I)*A(I)	00031500
C		00031600
C	LIM1 = INTEGER P	00031700
C		00031800
C		00031900
C		00032000
C	BETA TERMS	00032100
C		00032200
	MARKB = LIMA+1	00032300
	DO 61 I = 2,R	00032400
	S(MARKB,I) = ZERO	00032500
	IM1 = I-1	00032600
	DO 61 K = 1,IM1	00032700
	61 S(MARKB,I) = S(MARKB,I)+B(I,K)*A(K)**(MFOLD+LIM1-2)	00032800
	AA(MARKB)=PM1	00032900
	BB(MARKB) = MM(LIM1-2)	00033000
	IND2 = 1	00033100
	IND1 = MARKB	00033200
	LL = LIM1 - 4	00033300
	IF (LIM1 .EQ. 4) GO TO 66	00033400
	DO 65 K = 1,LL	00033500
	LL1 = INDEX(K+1)	00033600
	IPOW = LL-K+1	00033700
	DO 65 KK = 1,LL1	00033800
	IND1 = IND1 + 1	00033900
	IND2 = IND2 + 1	00034000
	DO 64 I = 2,R	00034100
	S(IND1,I) = ZERO	00034200
	IM1 = I-1	00034300
	DO 64 L = 1,IM1	00034400
	64 S(IND1,I) = S(IND1,I)+B(I,L)*S(IND2,L)*A(L)**M(IPOW)	00034500
	BB(IND1) = BB(IND2)*FACT(IPOW)	00034600
	65 AA(IND1) = AA(IND2)*PM1	00034700
	66 CONTINUE	00034800
	LL = LIMB - IND2	00034900

DO 68 K = 1,LL	00035000
IND1 = IND1 + 1	00035100
IND2 = IND2 + 1	00035200
DO 67 I = 2,N	00035300
S(IND1,I) = ZERO	00035400
IM1 = I-1	00035500
DO 67 L = 1,IM1	00035600
67 S(IND1,I) = S(IND1,I)+B(I,L)*S(IND2,L)	00035700
BB(IND1) = BB(IND2)	00035800
68 AA(IND1) = AA(IND2)*PM1	00035900
883 CONTINUE	00036000
JM3 = J - 3	00036100
GO TO (150,100,105,110,115,120,125,130,135),JM3	00036200
C	00036300
C CROSS PRODUCT TERMS	00036400
C	00036500
C	00036600
100 CONTINUE	00036700
C	00036800
C 5 TH ORDER TERMS	00036900
C	00037000
IND = 2*LIMIT(4)+1	00037100
CALL CROSS(LS,R,S,AA,BB,IND,2,2,0,0)	00037200
GO TO 150	00037300
C	00037400
C 6 TH ORDER TERMS	00037500
105 CONTINUE	00037600
IND = 2*LIMIT(5)+1	00037700
CALL CROSS(LS,R,S,AA,BB,IND,2,1,3,4)	00037800
GO TO 150	00037900
	00038000
C	00038100
C 7 TH ORDER TERMS	00038200
C	00038300
110 CONTINUE	00038400
IND = 2*LIMIT(6)+1	00038500
CALL CROSS(LS,R,S,AA,BB,IND,2,1,5,8)	00038600
CALL CROSS(LS,R,S,AA,BB,IND,2,3,0,0)	00038700
CALL CROSS(LS,R,S,AA,BB,IND,3,2,0,0)	00038800
CALL CROSS(LS,R,S,AA,BB,IND,4,2,0,0)	00038900
CALL CROSS(LS,R,S,AA,BB,IND,3,1,4,4)	00039000
GO TO 150	00039100
C	00039200
C 8 TH ORDER TERMS	00039300
C	00039400
115 CONTINUE	00039500
IND = 2*LIMIT(7)+1	00039600
CALL CROSS(LS,R,S,AA,BB,IND,2,1,10,18)	00039700
CALL CROSS(LS,R,S,AA,BB,IND,3,1,5,8)	00039800
CALL CROSS(LS,R,S,AA,BB,IND,4,1,5,8)	00039900

	CALL CROSS(LS,R,S,AA,BB,IND,2,2,3,4)	00040000
	GO TO 150	00040100
C		00040200
C	9 TH ORDER TERMS	00040300
120	CONTINUE	00040400
	IND = 2*LIMIT(8)+1	00040500
	CALL CROSS(LS,R,S,AA,BB,IND,2,1,21,40)	00040600
	CALL CROSS(LS,R,S,AA,BB,IND,3,1,10,13)	00040700
	CALL CROSS(LS,R,S,AA,BB,IND,4,1,10,18)	00040800
	DO 121 K = 5,8	00040900
121	CALL CROSS(LS,R,S,AA,BB,IND,K,1,K,8)	00041000
	CALL CROSS(LS,R,S,AA,BB,IND,9,1,5,8)	00041100
	CALL CROSS(LS,R,S,AA,BB,IND,2,4,0,0)	00041200
	CALL CROSS(LS,R,S,AA,BB,IND,2,1,46,48)	00041300
	GO TO 150	00041400
C		00041500
C	10 TH ORDER TERMS	00041600
C		00041700
125	CONTINUE	00041800
	IND = 2*LIMIT(9) + 1	00041900
C		00042000
	CALL CROSS(LS,R,S,AA,BB,IND,2,1,49,96)	00042100
	CALL CROSS(LS,R,S,AA,BB,IND,3,1,21,40)	00042200
	CALL CROSS(LS,R,S,AA,BB,IND,4,1,21,40)	00042300
	DO 126 K = 5,9	00042400
126	CALL CROSS(LS,R,S,AA,BB,IND,K,1,10,18)	00042500
	CALL CROSS(LS,R,S,AA,BB,IND,2,1,106,113)	00042600
	CALL CROSS(LS,R,S,AA,BB,IND,2,3,3,4)	00042700
	CALL CROSS(LS,R,S,AA,BB,IND,3,3,0,0)	00042800
	CALL CROSS(LS,R,S,AA,BB,IND,4,3,0,0)	00042900
	CALL CROSS(LS,R,S,AA,BB,IND,4,2,3,3)	00043000
	CALL CROSS(LS,R,S,AA,BB,IND,3,2,4,4)	00043100
C		00043200
	GO TO 150	00043300
C		00043400
130	CONTINUE	00043500
C		00043600
C	11TH ORDER TERMS	00043700
C		00043800
	IND = 2*LIMIT(10) + 1	00043900
C		00044000
	CALL CROSS(LS,R,S,AA,BB,IND,2,1,116,230)	00044100
	CALL CROSS(LS,R,S,AA,BB,IND,3,1,49,96)	00044200
	CALL CROSS(LS,R,S,AA,BB,IND,4,1,49,96)	00044300
	DO 131 K = 5,8	00044400
131	CALL CROSS(LS,R,S,AA,BB,IND,K,1,21,40)	00044500
	DO 132 K = 10,18	00044600
132	CALL CROSS(LS,R,S,AA,BB,IND,K,1,K,18)	00044700
	CALL CROSS(LS,R,S,AA,BB,IND,2,2,21,40)	00044800
	CALL CROSS(LS,R,S,AA,BB,IND,19,1,10,18)	00044900

CALL CROSS(LS,R,S,AA,BB,IND,20,1,10,18)	00045000
CALL CROSS(LS,R,S,AA,BB,IND,2,1,269,278)	00045100
DO 133 K = 46,48	00045200
133 CALL CROSS(LS,R,S,AA,BB,IND,K,1,5,8)	00045300
CALL CROSS(LS,R,S,AA,BB,IND,2,3,5,8)	00045400
CALL CROSS(LS,R,S,AA,BB,IND,2,2,46,48)	00045500
CALL CROSS(LS,R,S,AA,BB,IND,2,5,0,0)	00045600
C GO TO 150	00045700
C	00045800
C 12TH ORDER TERMS	00045900
C	00046000
C 135 CONTINUE	00046100
C	00046200
C IND = 2*LIMIT(11) + 1	00046300
C	00046400
C	00046500
C	00046600
CALL CROSS(LS,R,S,AA,BB,IND,2,1,287,572)	00046700
CALL CROSS(LS,R,S,AA,BB,IND,3,1,116,230)	00046800
CALL CROSS(LS,R,S,AA,BB,IND,4,1,116,230)	00046900
DO 141 K = 5,8	00047000
141 CALL CROSS(LS,R,S,AA,BB,IND,K,1,49,96)	00047100
DO 142 K = 10,18	00047200
142 CALL CROSS(LS,R,S,AA,BB,IND,K,1,21,40)	00047300
CALL CROSS(LS,R,S,AA,BB,IND,2,2,49,96)	00047400
CALL CROSS(LS,R,S,AA,BB,IND,19,1,21,40)	00047500
CALL CROSS(LS,R,S,AA,BB,IND,20,1,21,40)	00047600
DO 143 K = 41,48	00047700
143 CALL CROSS(LS,R,S,AA,BB,IND,K,1,10,18)	00047800
CALL CROSS(LS,R,S,AA,BB,IND,114,1,5,8)	00047900
CALL CROSS(LS,R,S,AA,BB,IND,115,1,5,8)	00048000
DO 144 K = 269,278	00048100
144 CALL CROSS(LS,R,S,AA,BB,IND,K,1,3,4)	00048200
CALL CROSS(LS,R,S,AA,BB,IND,3,3,2,2)	00048300
CALL CROSS(LS,R,S,AA,BB,IND,4,3,2,2)	00048400
CALL CROSS(LS,R,S,AA,BB,IND,3,2,20,20)	00048500
CALL CROSS(LS,R,S,AA,BB,IND,4,2,19,19)	00048600
CALL CROSS(LS,R,S,AA,BB,IND,2,4,3,4)	00048700
150 CONTINUE	00048800
C	00048900
C TEMPORARY INSERT TO CHECK VALUES OF AA AND BB COEFF	00049000
C	00049100
C LL = LIMIT(LIM1)	00049200
C IFAKE = 0	00049300
C DO 153 K = 1,LL	00049400
C IFAKE = IFAKE + 1	00049500
C IF (IFAKE .LT. 40) GO TO 153	00049600
C IF (KPRINT .EQ. 1) READ 497,II	00049700
C IFAKE = 0	00049800
C 153 PRINT 506,K,AA(K),K,BB(K)	00049900

C 506 FORMAT(4H AA(,I4, 4H) = ,D15.7,2X,4H BB(,

00050000

C 1 I4,4H) = ,D15.7)

00050100

IF (KPRINT .EQ. 1) PRINT 499

00050200

C
C
C

00050300

00050400

00050500

00050600

IF (LIMITS .EQ. J) JJ1 = J - 1

00050700

IF (OPTION .GT. 1 .AND. LIMITS .NE. J) GO TO 180

00050800

C

JJ2 = 1

00050900

JJ1 = J

00051000

C
C
C

00051100

EVALUATES FIRST T.E.C. OF ORDER J

00051200

SUM1 = UNITY/(AA(1)*P)

00051300

SUM3 = SUM1

00051400

DO 154 I = 1,R

00051500

SUM1 = SUM1 - C(I)*S(1,I)

00051600

SUM3 = SUM3 - CH(I)*S(1,I)

00051700

00051800

154 CONTINUE

00051900

SUM1 = SUM1 / BB(1)

00052000

SUM3 = SUM3 / BB(1)

00052100

IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3

00052200

IF (OPTION .EQ. 3) GO TO 155

00052300

PRINT 501,JJ1,JJ2,SUM1,SUM3

00052400

155 CONTINUE

00052500

C
C
C
C
C

00052600

00052700

EVALUATES T.E.C. FOR S(I,J) TERMS WITH ALPHA AS

00052800

LEADING COEFFICIENT (I .NE. 2)

00052900

00053000

00053100

IFAKE = 1

00053200

K = 1

00053300

KNT = 2

00053400

IPOW = LIM1 - 3

00053500

LIMD = INDEX(KNT)

00053600

156 CONTINUE

00053700

DO 159 KK = 1,LIMD

00053800

K = K + 1

00053900

SUM1 = UNITY/(AA(K)*P)

00054000

SUM3 = SUM1

00054100

DO 157 I = 1,R

00054200

SUM1 = SUM1 - C(I)*A(I)**IPOW*S(K,I)

00054300

SUM3 = SUM3 - CH(I)*A(I)**IPOW*S(K,I)

00054400

157 CONTINUE

00054500

SUM1 = SUM1 / (BB(K)*FACT(IPOW))

00054600

SUM3 = SUM3 / (BB(K)*FACT(IPOW))

00054700

IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3

00054800

IF (OPTION .EQ. 3) GO TO 159

00054900

JJ2 = JJ2 + 1

IFAKE = IFAKE + 1	00055000
IF (IFAKE .LT. 40) GO TO 158	00055100
IFAKE = 0	00055200
IF (KPRINT .EQ. 1) PRINT 499	00055300
158 CONTINUE	00055400
497 FORMAT(I3)	00055500
PRINT 501, JJ1, JJ2, SUM1, SUM3	00055600
159 CONTINUE	00055700
C	00055800
KNT = KNT + 1	00055900
LIMD = INDEX(KNT)	00056000
IPOW = IPOW - 1	00056100
IF (IPOW .GE. 1) GO TO 156	00056200
C	00056300
LIMD = LIMIT(LIM1) - LIMA	00056400
C	00056500
C	00056600
C	00056700
C	00056800
DO 162 KK = 1, LIMD	00056900
K = K + 1	00057000
SUM1 = UNITY/(AA(K)*P)	00057100
SUM3 = SUM1	00057200
DO 160 I = 1, R	00057300
SUM1 = SUM1 - C(I)*S(K, I)	00057400
SUM3 = SUM3 - CH(I)*S(K, I)	00057500
160 CONTINUE	00057600
SUM1 = SUM1 / BB(K)	00057700
SUM3 = SUM3 / BB(K)	00057800
IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3	00057900
IF (OPTION .EQ. 3) GO TO 162	00058000
IFAKE = IFAKE + 1	00058100
IF (IFAKE .LT. 40) GO TO 161	00058200
IFAKE = 0	00058300
IF (KPRINT .EQ. 1) PRINT 499	00058400
161 CONTINUE	00058500
JJ2 = JJ2 + 1	00058600
C	00058700
556 PRINT 556, K, LIM1, AA(K), P, PP1	00058800
FORMAT(2(2X, I4), 3(2X, D15.7))	00058900
PRINT 501, JJ1, JJ2, SUM1, SUM3	00059000
162 CONTINUE	00059100
IF (KPRINT .EQ. 1) PRINT 499	00059200
499 FORMAT(//)	00059300
180 CONTINUE	00059400
182 CONTINUE	00059500
TERROR = DSQRT(TERROR)	00059600
RETURN	00059700
END	

```

SUBROUTINE CROSS(LS,R,S,AA,BB,INDEX,TERM1,POWER,TERM2,
1  TERM3)
  INTEGER TERM1,TERM2,TERM3,POWER,R
  DOUBLE PRECISION S(LS,R),AA(LS),BB(LS),FACT(7)

C
  DATA FACT/1.0D0,2.0D0,6.0D0,24.0D0,
1  120.0D0,720.0D0,5040.0D0/

C
C
C

  IF (TERM2 .EQ. 0) GO TO 20

  COMPUTES S(INDEX+J,I)=S(TERM1,I)**POWER * S(TERM2+J,I)
    FOR J=0,1,...,TERM3-TERM2

  KK = -1
  DO 10 K = TERM2,TERM3
    KK = KK + 1
    INDX = INDEX + KK
    AA(INDX) = AA(TERM1)**POWER * AA(K)
    IPOW = POWER
    IF (TERM1 .EQ. K) IPOW = IPOW + 1
    BB(INDX) = BB(TERM1)**POWER * BB(K) * FACT(IPOW)
    DO 10 I = 2,R
10  S(INDX,I) = S(TERM1,I)**POWER * S(K,I)
    INDEX = INDEX + TERM3 - TERM2 + 1

C
C

  RETURN

C
C  COMPUTES S(INDEX,I) = S(TERM1,I)**POWER

20  CONTINUE
    AA(INDEX) = AA(TERM1)**POWER
    BB(INDEX) = BB(TERM1)**POWER * FACT(POWER)
    DO 25 I = 2,R
25  S(INDEX,I) = S(TERM1,I)**POWER
    INDEX = INDEX + 1

C
C

  RETURN
END

```